SlideTalk Voice Component Implementation

by

SANJEEV KOHLI

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
APRIL 1997

SlideTalk Voice Component Implementation

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Sanjeev Kohli

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1997

Certificate

This is to certify that the work contained in the thesis entitled **SlideTalk Voice Component Implementation** by **Sanjeev Kohli** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree

Dr D Manjunath
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Kanpur

Dedicated to my wife, who enthused me all along this work, and to my parents whose efforts and sacrifices have enabled me to reach this stage

Acknowledgements

First of all I would like to express my sincere gratitude to my thesis supervisor Dr D Manjunath for his invaluable guidance and constant encouragement throughout the work I would like to thank him for giving me complete freedom in work I am very thankful to him for giving me an opportunity to undertake this challenging topic as my M Tech thesis It was his wonderful association that enabled me to achieve the goal of this work

Special thanks to Manoj, Sumeet Alkesh, Deepak Pankaj Jayesh and Mohit for their help and constant encouragement I thank all my friends especially those belonging to Doordarshan and the Telecom group for making my stay at IITK a memorable one I would also like to thank all the members of the Telematics group especially Mrs Sandhya Sule for her constant help and guidance

Last but not the least, I would like to thank all the IITK community for providing me with a peaceful ambience and a home away from home

Sanjeev Kohli

Abstract

Proliferation of personal computers explosive growth of internet the changes in communication services and availability of inexpensive hardware on desktop has paved its way for group collaboration in distributed environments Many computer tools have extended their support for effective group collaboration on Unix platform

In this thesis we have developed a PC based teleseminaring tool SlideTalk on MS Windows platform SlideTalk allows the speaker and the audience to interact during the conference through PCs networked either on the same LAN or on an internet The system software is designed completely under ObjectWindows environment and uses IP group delivery model for communication between members which works on top of UDP as the underlying transport protocol

The complete implementation work for SlideTalk has been done in two parts Display of Slides and Voice Conferencing This thesis work deals with the voice conferencing component The speaker s voice is multicast in real time to all the group members The complete development of SlideTalk has been done on top of WhiteBoard which provides the functionality to exchange graphics and multifont text in a multicast environment

Contents

List of Figures

Chapter 1

Introduction

Relatively inexpensive hardware for processing audio and video data are rapidly becoming available for desktop Personal Computers At the same time greater network bandwidths at lower prices is making its way to the desktop These developments have stimulated interest and demand for distributed applications that require these new capabilities for processing the streams of digitized audio and video data For example applications such as computer based conferencing multimedia mail and document processing have been developed to enhance users work environments and to foster a more effective group collaboration possibly in a distributed setup

In this thesis and its companion thesis [1] we have developed a remote seminaring tool SlideTalk for the PC platform In the following sections we briefly introduce the features of SlideTalk the existing implementation and the features that we have added to enable SlideTalk to become a useful seminaring tool

1 1 SlideTalk

The goal of SlideTalk is to enable tele seminaring The seminar session is delivered with the speaker and the audience interacting through PCs that are networked either through the same LAN or through an internet A graphical user interface (GUI) provides them with a user friendly environment during the seminar

In SlideTalk all the information is multicast Multicasting is used to communicate among a group and the members of the multicast group are identified by a unique address known as *group address*. (Network nodes have one or more multicast addresses in addition to their unicast addresses. The multiple multicast addresses correspond to memberships in multiple groups.) Thus, before commencing a session both the speaker and the audience will decide upon a group address to which their session will belong and the time at which the seminar is to begin. To join the seminar session the speaker or audience first submit the group address to the application to get a window of SlideTalk.

The SlideTalk window of the speaker and of the audience will offer the same menu choices, but the speaker will control the utilization of some of these choices. The menu choices available are: slide (to display a PostScript slide in the SlideTalk window), voice (to begin or end a one way voice communication), graphical annotating objects (to either annotate on a slide or to draw figures on a blank screen), and text (that allows a participant to type multifont, multicoloured text). All information generated at a participant's machine is multicast to all the group members. This means that all the members of the group will always have the same information displayed in their windows at all times.

The slide feature allows importing of a PostScript slide file from the Ghostview application into the SlideTalk window. This import information will then be multicast so that the same slide is opened in the SlideTalk window of all the group members. It is assumed that all members of the group have the slide that is to be imported in the same path as in the speaker's machine. For voice communication, the speaker's talk is multicast in real time.

The speaker controls the slide display and voice communication. If the speaker opts for the slide, no member in the audience would be able to open a slide. Similarly, if the audio session is started by the speaker, the microphone of the audience is deactivated.

The graphical objects supported by the SlideTalk are: ellipse, rectangle, straight line, and curved line. It also has a text editor. For text the font type, font size, and font colour can be selected. The graphics editor has a provision to select the colour and border thickness of the figures. The text and figures can be drawn either on the blank pages or on the slides to annotate during the seminar.

Only one menu item is accessible at any time except for audio. One other option is supported concurrently with audio to enable the speaker to continue talking while importing a slide or while annotating on it.

The SlideTalk session data is multicast over the network. Since the data is transmitted only once to all the participants, the bandwidth requirement on the network is not a function of the number of participants and SlideTalk can support any number of participants simultaneously. Participants can log into or log out of the session at any time without disrupting the ongoing session. The member who joins late can receive only the data which has been transmitted after the joining time.

1.2 Previous Implementation

The design and implementation of the WhiteBoard was done by Gupta and Mukhopadhyay [2]. They developed the WhiteBoard in which they provided the basic functions for graphics and text. They used multicast for communication among the group members. The packetization of the data is handled by the application. The communication mode is free mode, which in this case means that there is no limit on the number of members interacting at the same time.

WhiteBoard also has options for font style, font size, and font colour. As soon as some text is typed on any of the consoles, a packet is made on a per character basis filling the fields like the character value, the coordinates of the character and the font data. Then the packet is multicast. Similarly if some graphical object (a curved line, a straight line, a rectangle or a circle) is drawn, a packet is filled with information like type of the object, the pen size, the border thickness, coordinates of the figure and the colour of the figure drawn.

The design of the WhiteBoard is object oriented. Separate layers have been designed to interface with Windows Sockets, preparation and transmission of packets and starting of the session. The highest abstraction allows the user to draw a graphical object or write text on the WhiteBoard.

Since the design of the WhiteBoard is object oriented, it is an excellent platform to build SlideTalk on top of it. We have added features to it by modifying its top layers while the lowest laver that deals with multicasting of the information is retained from WhiteBoard Thus, we have enhanced the features of the WhiteBoard to realize SlideTalk

1 3 Our Implementation

In this implementation we have incorporated the two features of slide display [1] and voice communication Primitive forms of audio and video playback have also been incorporated PostScript slide files can be imported into the window of SlideTalk Other slide formats like the PowerPoint bitmap gif could also be displayed using a similar technique but has not been implemented The PostScript files cannot be displayed directly in the window and requires the Ghostview application for displaying it Thus to display a slide the same slide is opened in the Ghostview application and then imported into the display window The main challenge is to import the data of one application to the other without loss of resolution When a slide gets displayed on the speaker s console the same slide file information is multicast to the group Using the graphics and text functions developed for the WhiteBoard the speaker can annotate the slide

In the voice component we transmit voice in WAVE format with 8 bit PCM encoding The audio buffers are filled by the Sound Blaster audio device driver and packetized voice is multicast The mode of interaction in the voice mode is FIFO speaking requests are serviced in order of arrival but only one microphone is enabled at a time Here the main challenge is to minimize latency and loss of fidelity and also to keep the voice session synchronized

1 4 Organization of the Thesis

In this thesis and its companion the two main features of SlideTalk - slideshow and voice communication have been developed The rest of the chapters of this thesis have been organized as follows Chapter 2 describes in detail the WhiteBoard platform on which SlideTalk was developed Chapter 3 deals with the design issues for the speech communication component and the implementation details Chapter 4 discusses the testing of the implementation and also contains concluding remarks and future work

The work of implementing of import of slides and of voice trasnsmission was necessary to make SlideTalk a usable seminar tool To avoid duplication of effort Chapters 1 and 2 of this thesis and the companion thesis [1] are common

Chapter 2

The Previous Implementation - WhiteBoard

The WhiteBoard implementation of Gupta and Mukhopadhyay [2] uses a multicast framework and features exchange of graphics and text information. The motivation for the WhiteBoard design and features is from the work of Van Jacobson [3]. Van Jacobson et al developed a framework for scalable reliable multicast (SRM). The functionality provided in SRM is the eventual delivery of all data to all group members without enforcing any particular order. Their framework has been prototyped in wb, a distributed teleseminaring application which maintains a shared window supporting graphics and text. This application works under X-windows.

The framework of the WhiteBoard borrows heavily from the SRM work. As in SRM, WhiteBoard ensures eventual delivery of all the data to the entire group. The specification of the WhiteBoard was such that global ordering of information delivery was not necessary. Since WhiteBoard was not designed to be a distributed conferencing package, the requirements on real time delivery were also not very stringent. The design however permits an application that is related to it to incorporate its own ordering on the packets exchanged in the group.

2.1 The Framework for WhiteBoard

The communication model for WhiteBoard follows the IP group delivery model. In IP multicast, corresponding to every active session, there is a group address. Data sources simply write to the group address and receivers gather data by listening to the group address. Individual members do not need to know the number of active senders at any instant or the IP address of other members. This model adds functionality to the IP model to ensure that the shared window is consistent among the members and that the members exercise some kind of ownership over the data they create. UDP [4] has been chosen as the underlying transport protocol because the IP group delivery model works only on top of UDP. Since there is no client server mode of communication, no centralized ordering of the objects is possible.

The design is object oriented [9] and consists of classes like client, shape, frame and socket. The shape class is further subdivided into straight line, rectangle, ellipse and the curved line. This class is concerned with the display of the shape data on the window. The client class maintains the multicast session and involves handles of other classes (frame and socket) for handling the task of receiving packets and their transmission. The frame class is concerned with packet formats and defines their fields. The socket class deals with the raw data on the network by opening a socket.

The application is structured in a layered manner: the Shape layer, the Frame layer and the Bytestream layer. The Shape layer interprets data in terms of *shapes* and maintains the objects displayed in the window. The Frame layer is concerned with the generation of the application protocol frames and communicates with the peer layer to maintain a session. The Bytestream layer operates at the socket level [6] and interprets data in terms of bytes. The detailed description of the functions of layers is explained in a later section.

The application level protocol is used by the group members to force a consistent interpretation of the data. A shape can be thought of as an object with the attributes of Type, Identification, Pen Size, Colour and the Position on the screen. The protocol interface to the Shape layer is in terms of objects with the above attributes.

The IP group delivery model works on top of UDP. Hence the protocol assumes lossy

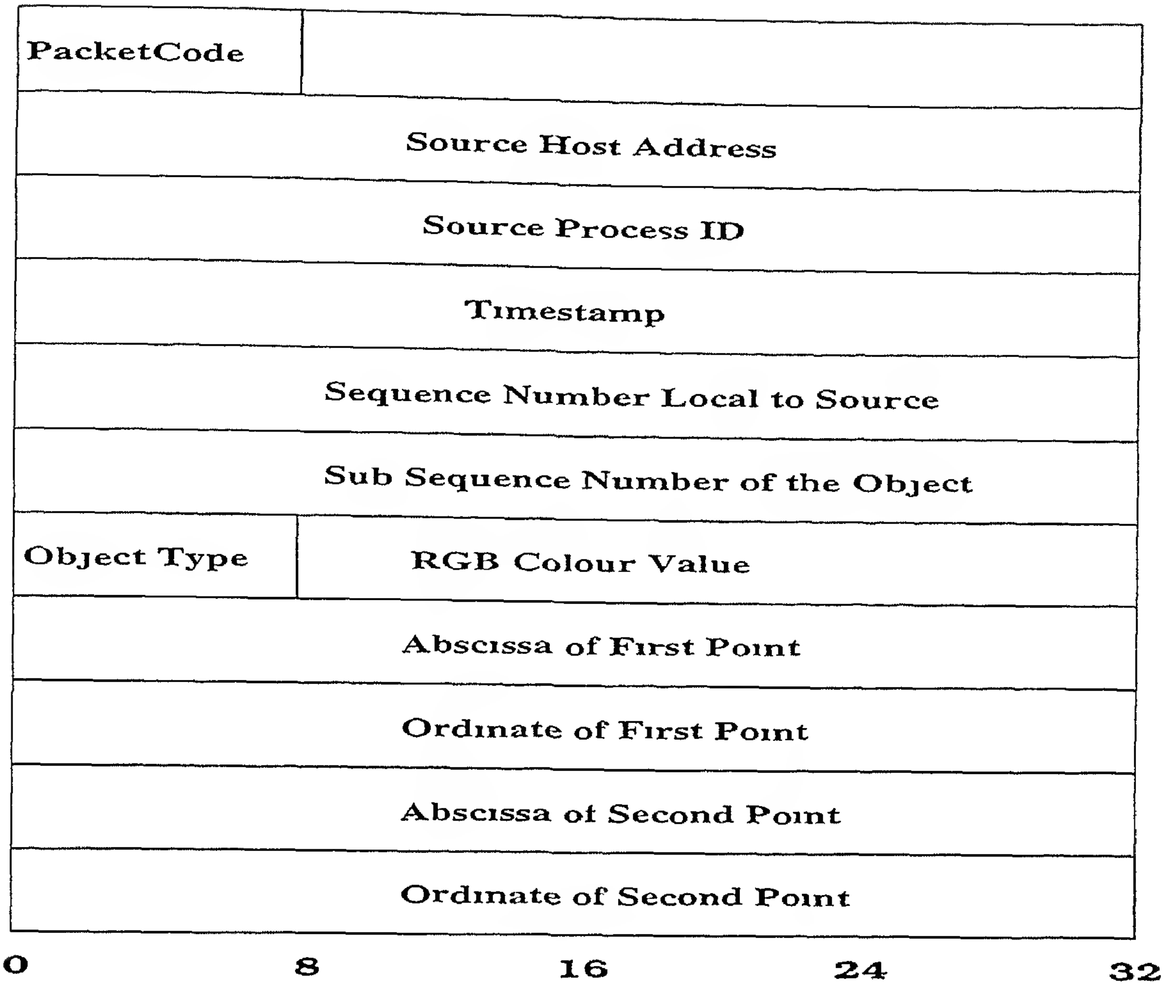

Figure 2 1 The Packet Structure for Graphical Object

connectionless service from the underlying network Packets are multicast to the group whenever an active member puts up a shape in his window At the same time all the other members keep listening to their *group socket* and read the data whenever it is available

The packets are of fixed size (64 bytes) This choice of packet size does not waste too much bandwidth The data and control information for all the shapes is contained within 64 bytes There is a separate packet format for graphics objects and text These two packet formats are shown in Figure 2 1 and Figure 2 2 In the packet various fields have been defined The object type field differentiates between various objects The rest of the fields define the attributes of the object based on which the object can be properly ordered and displayed in all the peer windows

Packet Code			
Source Host IP Address			
Source Process ID			
Timestamp			
Sequence Number Local to Source			
Sub Sequence Number of the object			
Object Type	RGB Colour Value		
Height of Character			
Width of Character			
Escapment of Character			
Orientation of Character			
Weight of Character			
Italics	Underline	Strike Out	Char Set
O/P Precision	Clip Precision	Quality	Pitch
Abscissa of Location			
Ordinate of Location			
Char Value			

0 8 16 24 32

Figure 2 2 The Packet Structure for Text Object

Each user level object is uniquely identified by the user s Host IP address user s process ID and a sequence number local to the user The packets describing the same object are distinguished and ordered by the sequence number The construction of objects is controlled by passing some control information in the packets In the case of a straight line, a rectangle, or an ellipse since the object needs to be displayed only when its position has been entirely decided, a single packet is needed with the corresponding control code which has the two relevant points (these points denote the two end points in case of a straight line and in the case of ellipse and rectangle they denote the left-top and right bottom corners of the rectangle) However, in the case of the curved line the process of drawing should be visible on all the windows To achieve this a control code denotes the start of a curved line The points comprising the curved line are tagged with the sequence numbers so that they can be indexed properly even if they arrive out of order. Finally, after the last point has been sent, a special packet denoting the end of the curved line is multicast to the group. Text is sent as a packet for each character along with the relevant font information.

2.2 WhiteBoard Design Issues and Implementation

One major issue in the design of the WhiteBoard was to provide for concurrent update in any part of the window. Any method of overcoming the concurrency problem must necessarily identify the packets uniquely and provide the queueing of the packets so that the objects are displayed in the window at the appropriate time. Two major problems in incorporating this during multicast are the reflected packet problem and the simultaneous write problem. Due to the asynchronous mode of communication, each packet sent to the network is immediately received back. This is termed as the reflected packet. This leads to the shift of control to another message handler. This message handler then identifies the packets to be its own and drops them. A similar problem occurs when two members write simultaneously. At the uppermost layer this implies the presence of more than one Graphics Device Context at the same time. For each received message the WhiteBoard allocates a device context, updates the window using the message and then deallocates the device context. For the local graphics, however, a separate device context is maintained throughout the creation of the shape.

The ordering requirement for the WhiteBoard is not very stringent. The requirement is that the windows of different members be consistent, more or less. However, the local ordering is done so as to allow for the unique identification of each object. The local ordering is achieved by providing the objects with the sequence numbers. Each host orders the objects for display depending on the order of their arrival. A cross linked chain of host list and the shape list achieves this order. In the cross linked structure it is assured that the objects corresponding to a particular host are in order, even if they have arrived out of order. Thus, the only inconsistency that can be present among different windows is regarding the order of placement of the objects created by the different members.

2 2 1 The Layers

The Bytestream Layer

This layer communicates directly with the network To make this layer an independent identity, a SocketBase class has been introduced This provides the upper layers with a handle to start one end of a session without being concerned about the socket options the network errors and maintenance The class Multicast SocketBase is derived from the SocketBase class and it provides multicast support The upper layers can start multicast sessions and perform network read or write operations using these layers Further the support for reading and writing is entirely asynchronous and the parent window receives the messages that contain information regarding the connection (socket) that has the message to be read

The Frame Construction Unit

WhiteBoard has a separate frame construction unit that deciphers the packets received from the network This unit is organized in such a way as to minimize the overhead on the other layers, and also to speed up the job This unit comprises both encapsulation of the outgoing data into frames as well as the retrieval of data from the incoming packets Outside this unit the treatment of frames is in terms of abstract entities that form part of the frames In essence this means that the frame construction unit understands all packet formats (and only this unit needs to be aware of all the packet formats) and the upper layers get the required information from this layer This allows the unit to hide the conversion of data into network byte order before placing it into the frame and conversion of data extracted from an incoming frame into the local byte order before giving it to the unit that needs the data This conversion is necessary to communicate with the WhiteBoard applications running on the platforms that support different byte orderings

The Frame Layer

The frame layer primarily consists of the class Client which derives from the Multicast SocketBase class and extends its functions with a session maintenance functionality for the WhiteBoard The frame layer provides a handle to the upper layer which mentions only the shape to be written to the network This layer calls the frame construction unit to generate the frame and calls the handle of the bytestream layer to transmit the packet The incoming packets are taken from the Bytestream layer The frame layer constructs the shapes from these packets and returns them to the upper layer

The frame layer also does the work of session maintenance All the queues and lists of shapes are maintained here and this layer performs the memory management for the WhiteBoard by maintaining the queues

The Shape Layer

This is the layer that provides the highest level of abstraction and deals with shapes that are put up on the window and the operations pertaining to them An abstract class called *shape* has been designed and from it the specific classes like straight line rectangle ellipse curved line and text have been derived The shape layer passes on the shape objects to the frame layer which in turn packetizes them and sends them over the network through the Bytestream layer At the receivers end the frame layer passes pointers to the shapes to the shape layer which then displays them by calling the appropriate display function for the shape

2 3 Conclusion

Using layers and functions defined for them a modular structure of the WhiteBoard was developed by Gupta and Mukhopadhyay This design makes it easy to incorporate new features or enhance existing features in the WhiteBoard because only one or two modules (layers) may have to be restructured with no changes in any other module

We have used WhiteBoard as a platform for developing SlideTalk In order to incorporate slide import and voice transmission features we have modified the top layers while the lower layers that take care of multicasting of data are retained from WhiteBoard

Chapter 3

Design and Implementation of the Voice Component

Our emphasis in this thesis work is on supporting real time communication to enable desktop conferencing on Personal Computers. The approach that we can adopt for the real time communication of the voice data depends on numerous factors like the architecture of the audio subsystem, the interaction of the operating system with the audio subsystem, the available network bandwidth, and the degree of MAC layer support in the network for real time communication.

In the present thesis work of incorporating the voice capability in SlideTalk, we first discuss the design issues to be considered for supporting voice conferences. Thereafter, we describe our software oriented approach for implementing voice communication facility and integrating it into SlideTalk.

3.1 Design Issues for Real-Time Audio Conferencing

The most stringent conditions for a high quality of presentation in a real time communication are presented by voice data. This is because there is very little temporal coherence [12] in the voice data, which means that if any packet is missed or gets played a multiple number of times it is easily noticed by the user. In such a scenario, the design for voice conferencing system requires a study of all the aspects concerned with it.

the MAC layer support, the computational requirements and the buffer management strategies

3 1 1 MAC Layer Support

More important than the available bandwidth are the restrictions that multimedia applications, in particular interactive distributed multimedia applications impose on the underlying network There are local area networks [15] which have a low bandwidth but do not support any priority for its traffic and may be unfair in the distribution of bandwidth In high load situations they exercise no control over access delay or the available bandwidth per application On the other hand are MAC layer protocols that are fair in distribution of bandwidth have a large bandwidth and can set priorities for their traffic (e g FDDI) Some of them also support synchronous traffic by guaranteeing bounds on the access delay

In an asynchronous network environment (e g Ethernet) the way the network is used by others cannot be controlled and therefore the delay and the number of discontinuities that will occur over any interval of time cannot be bounded This can lead to jitter in the voice communication and/or packet loss These two issues are discussed below

Jitter

In order to sustain a high quality conference samples of voice data must arrive at the receiving station so as to be played at a time that is at a constant offset from the time at which they were generated Although these samples can be generated at the desired rate by the audio hardware the exact rate at which the samples arrive at a receiver can be grossly distorted by poor operating system scheduling on the transmitting machine and varying load in the network The problem is to ameliorate the effect of jitter (i e variance in the packet inter arrival time at the receiver) in the arrival stream

Jitters in a transport system takes place due to the variations in latency For each audio packet, the total latency will have the following components

- the time spent in the audio device (e g Sound Blaster) pipeline at the originating machine
- the time spent at the network interface waiting to access the network

- the physical network transmission time

- the time spent queued at the receiver waiting to get played out

The main factor in the control of latency is the network interface time Depending on the underlying network (MAC layer) the jitter can be either bounded or unbounded Bounded jitters (upper bound on delay) are experienced on local area networks like FDDI which support synchronous traffic with the delay limit configurable at the network initialization time On Ethernet we observe unbounded delays Network traffic bursts increase latency and thus introduce jitters at the receiving workstation

The effect of jitter can be reduced by using an initial playout delay Let τ be the playout delay measured as the time between the instants at which the first sample was generated and that at which it was played out Large values of τ can absorb large values of jitter However a large value of τ affects real time communication especially when there are more than one involved in communication Thus there is a trade off in the value of τ that can be chosen to minimize the effect of jitter and to allow real time communication

However if the packets are generated on one LAN and received on another LAN via internet then even if both the LANs offer bounded delay unbounded jitter may still be experienced This is so because the underlying UDP/IP does not guarantee bounded delay

Packet Loss

As the load on the network increases packets may be lost as they pass through routers or bridges Traditional data communication strives to provide reliable end to end communication between two peers Existing communication systems always use checksum and sequence numbering for error control and some form of negative or positive acknowledgment with packet retransmission for error recovery If checksum is not performed either in hardware at the media access control (MAC) or the link layer it can affect the system performance The (negative) acknowledgment with subsequent retransmission handshake adds more than a full round trip delay to the transmission of the data For time critical data of multimedia or conferencing application the retransmitted message might thus be altogether useless

The issue of reliability thus becomes even more complex for multipoint communi cation Currently there exists no generally agreed upon scheme for the semantics of a reliable multicast service Thus networks simply provide unreliable multicast and leave the implementation of reliability to the higher communication layers usually at the application

3 1 2 Computational Requirements

It is quite essential to make a proper choice of audio processors as it is a vital factor for determining the load on the CPU when a conferencing application is going on At one extreme are the systems where digital or analog audio signals are acquired by largely autonomous audio processors that are capable of delivering the digital data directly to an assigned buffer or device with little if any intervention by the CPU Examples of such a system are those with direct attachment of off the shelf teleconferencing codecs to workstations At the other extreme are computer systems where the CPU is intimately involved in the control of the audio processors and in the communication of the data between the processes controlling the audio processors An example of such a system is the Sound Blaster 16 bit Audio Record system In this system the CPU at the sender s end directs the allocation of a buffer supplies the buffer to the audio subsystem collects it from the subsystem and then transmits it While at the receiver s end the CPU receives the buffer supplies it to the subsystem which then plays it out and then returns the buffer to the application for initializing The effective use of such an audio subsystem requires that real time services be provided by the operating system

3 1 3 Buffer Management

The trickiest data to handle for transmission is voice data and it requires proper buffer management to handle it A technique that can be used for transmitting and receiving voice data is the double buffering method For reception one buffer is used to store the incoming data and the other buffer is used for playback Once the first buffer is filled or the audio playback of the second buffer has completed the two buffers will be switched The buffers for transmission of voice are used in the similar manner Here again, two buffers are used for transmission of sound One buffer is used to store voice data captured by the application. When it is full, the voice data is sent to the multicast address. Meanwhile the other buffer is supplied to the audio device to store the voice data.

There is, however, a drawback associated with the double buffering method. If the variance in latency is high such that it exceeds the packet generation time, in such a case there is a high probability that packets will be lost. This is so because if two consecutive packets experience large latency and the following packet observes small latency, it might reach when both the buffers of receiving station are busy. In such a scenario that packet will get dropped.

3.2 Implementation of the Speech Component

In this thesis we have implemented Voice Support for the SlideTalk application. For voice digitization and play out we use Sound Blaster 16 Bit Audio Record hardware and Pentium PCs. We assume that the network supports multicast UDP/IP. Our work is based on the object oriented framework of WhiteBoard. The lowest layer in WhiteBoard deals with raw data and supports multicasting. We therefore could deal directly with the audio issues and packet format for its transmission without worrying much about the network read or write operations. In the upper layers new objects have been designed. In the Shape layer a new "shape" called Voice has been developed and in the Frame layer a new packet format has been incorporated which supports all the required fields for this new shape. The packet structure is as shown in Figure 3.1.

The mode of communication is FIFO (first in first out). In this mode, the first person opting to speak would get the chance and the speech menu of the other members of the group would get deactivated till this speaker completes talking. The mode has been selected as FIFO because our audio subsystem requires intimate involvement of the CPU in real time for its control as well as for communication of the audio data between the processes controlling the audio processors. In such a scenario if more than one speaker is active at a time, it severely increases the CPU load.

In the FIFO mode if two speakers opt to speak at the same time, both would be intimated of the collision and as a result they would be required to opt out of the audio

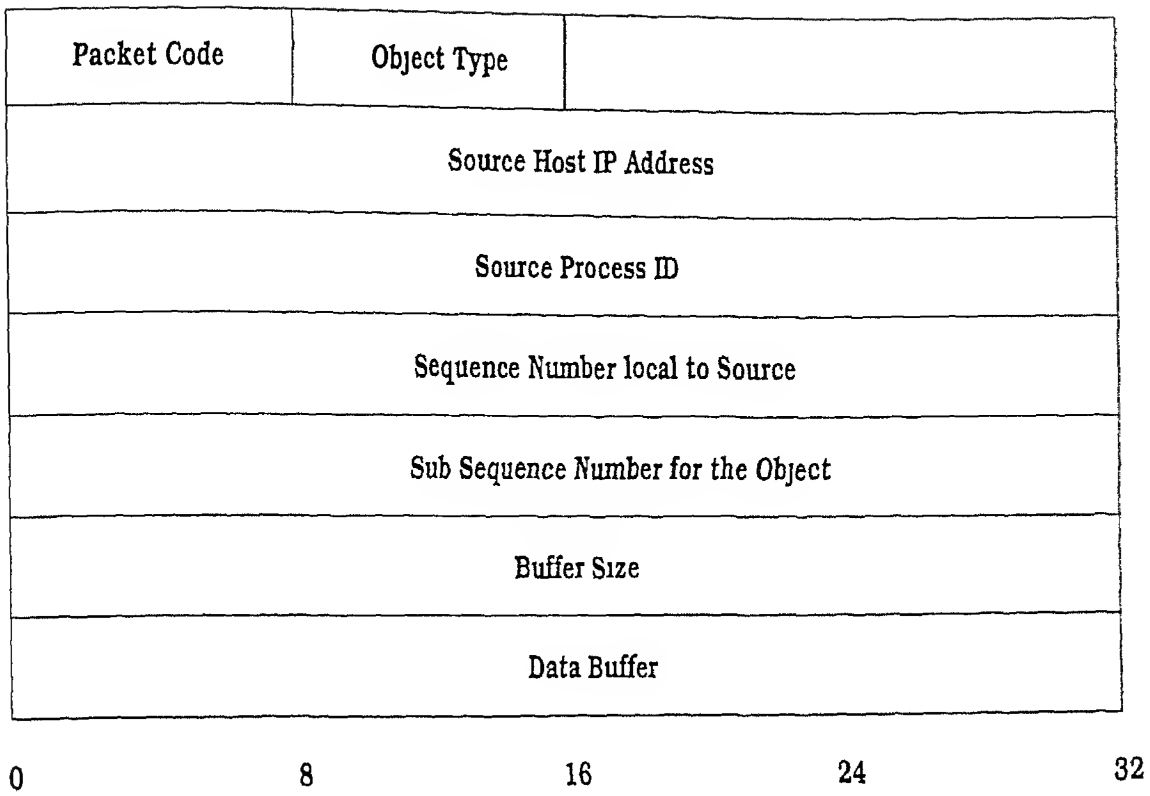

Figure 3 1 The Packet Structure for Voice Data

mode and try after some time so that the collision gets resolved

When the speaker opts for voice communication a message is given to the WAVE device (the Sound Blaster card) to open it in the record mode The WAVE device of the speaker is supplied the WAVEFORMAT The WAVEFORMAT structure is as shown below

```
typedef waveformat_tag{
 WORD wFormatTag
 WORD nChannels
 DWORD nSamplesPerSec
 DWORD nAvgBytesPerSec
 WORD nBlockAlign
 }WAVEFORMAT
```

wFormatTag specifies the format type of audio encoding nChannels indicates the number of audio channels nSamplesPerSec specifies the number of samples to be generated per second. nAvgBytesPerSec specifies the average data transfer rate in bytes per second. nBlockAlign is the minimum atomic unit of data, i.e. one sample size.

We have chosen 8 bit linear encoded PCM encoding with the mono recording. Each sample is, therefore, one byte and we collect 10,000 samples per second. Thus SlideTalk generates a traffic of 10Kbytes per second for supporting the voice conferencing.

After the device is opened, the buffers are supplied to it to record the voice data. The opened WAVE device accepts data only in the WAVEHDR format, which in turn points to the buffer space for the data. The WAVEHDR structure is as shown below.

```
typedef struct wavehdr_tag{
LPSTR lpData;
DWORD dwBufferLength;
DWORD dwBytesRecorded;
DWORD dwUser;
DWORD dwFlags;
DWORD dwLoops;
struct  wavehdr_tag *lpNext;
DWORD reserved;
}WAVEHDR;
```

lpData points to the waveform data buffer. dwBufferLength specifies the length of the data buffer. dwBytesRecorded specifies the amount of data recorded in the buffer (for the input device). dwUser specifies 32 bits of user data. dwFlags provides information about the data buffer. dwLoops specifies the number of times to play the loop. The next two fields are reserved for future use.

Before a WAVEHDR data is given to the WAVE device, it is prepared for the record (in) mode and then transferred to the buffer of the WAVE device. When the WAVE device fills the supplied buffer, the flag in the WAVEHDR is set. Meanwhile the application keeps looking at this flag; when it finds the flag set, it empties the buffer associated with that WAVEHDR and packetizes its contents.

Similarly at the receiver's end, when the indication of the start of the session arrives the WAVE device is opened in the playout mode. Again a WAVEFORMAT is given to the WAVE device mentioning the sample size and the sample rate at which we want to play the audio received

For a good response of the received voice it is essential that the WAVE device in the playout mode be opened with the same specifications as that of the sender s In the playout mode the WAVEHDR is filled with information like the pointer to the buffered data received and its size Then it is added to the WAVE device WAVE device informs the application when it has played the voice data supplied to it with the help of a flag

The trickiest thing in the voice conferences is the manner in which the voice data is transmitted This is so because we are working with asynchronous packet switched networks (Ethernet), with no MAC layer support for real time communication Ideally the voice data should be transmitted as soon as it gets generated When the network access time to send a frame becomes large than the frame generation time queues form in the transport system The length of these queues and the policies for managing them can strongly affect the throughput and the latency of the conference For example if a buffer cannot be transmitted when it is generated enqueueing the buffer only serves to increase its latency However if frames that cannot be transmitted immediately are discarded the throughput of the conference may unnecessarily decrease (with a corresponding increase in the number of discontinuities) A transport queue of length greater than one is useful only if the additional latency it introduces can be tolerated by the users of the system

The voice stream is not as susceptible to the effect of queue length because for the network that we are using a large number of audio samples can be transmitted in a single packet We use only two buffers which means that we can get a queue of at the most one at any time This choice for two buffers has been made because our audio buffer (1000 bytes) takes only 0 8 ms for propagation and we are transmitting just one packet per 100ms thus there is sufficient time to send a buffer while the other buffer is getting filled up Hence unless very high load conditions persist for long periods on the network our conference can sustain maintaining high fidelity, and observing very few discontinuities Should the bandwidth available to the conference decrease below that required for the full audio sampling rate, more aggressive application level mechanisms such as changing the coding or compression scheme will have to be employed to sustain the conference

While transmitting, one buffer is used to store voice data captured by the microphone. When it is full, the data is attached with a header and then sent to the multicast address. At this time, the other buffer is attached with the WAVE device to store the voice data. Thus the two buffers keep switching their roles. The flow diagram of the scheme is as shown in the Figure 3.2.

In our implementation, we do not introduce any initial playout delay. We packetize this data by attaching a header of 28 bytes to it and then multicast it. The size of the packet has been chosen so that on one hand it does not contain a large amount of speech data so as to cause severe discontinuity in case of a packet loss, and on the other hand it is not small enough to require network access more frequently (which increases the probability of packet delay/loss) and increases the CPU load to transmit the buffers. In the work done by Jeffay *et al* [13], they recommend redundant audio data transmission with packet size as large as 10Kbytes to support a high fidelity audio and video conference. The use of this scheme drastically reduces the chances of packet loss and the discontinuities but comes at the cost of higher bandwidth requirement.

The effect of the playout delay is to cushion the conference from the jitter caused by the bursty network traffic. In our implementation, if a network traffic burst of upto 100 ms occurs in the network, SlideTalk will not incur any packet loss.

For reception of the voice data, we again use two buffers. Packets arriving from the network are processed by the Frame Layer. Loopback packets are screened out whereas the packets from the other hosts have their headers extracted. The header gives the information like the speaker's IP address and the process ID. After stripping off the header, the voice data is copied to the buffer deployed for storing the data. When the buffer is full, it is submitted to the audio output device. The buffer previously submitted to the audio device is now used to store the incoming packet. When the audio device has finished processing the buffer whose address is pointed to by the WAVEHDR, the device sets the relevant flag of WAVEHDR. The main program processes the buffer after it sees the flag set. The buffer that currently stores the incoming packet is then submitted to the audio device and the buffer returned by the audio device takes its place to store the incoming packet instead. The flow diagram for this scheme is shown in Figure 3.3.

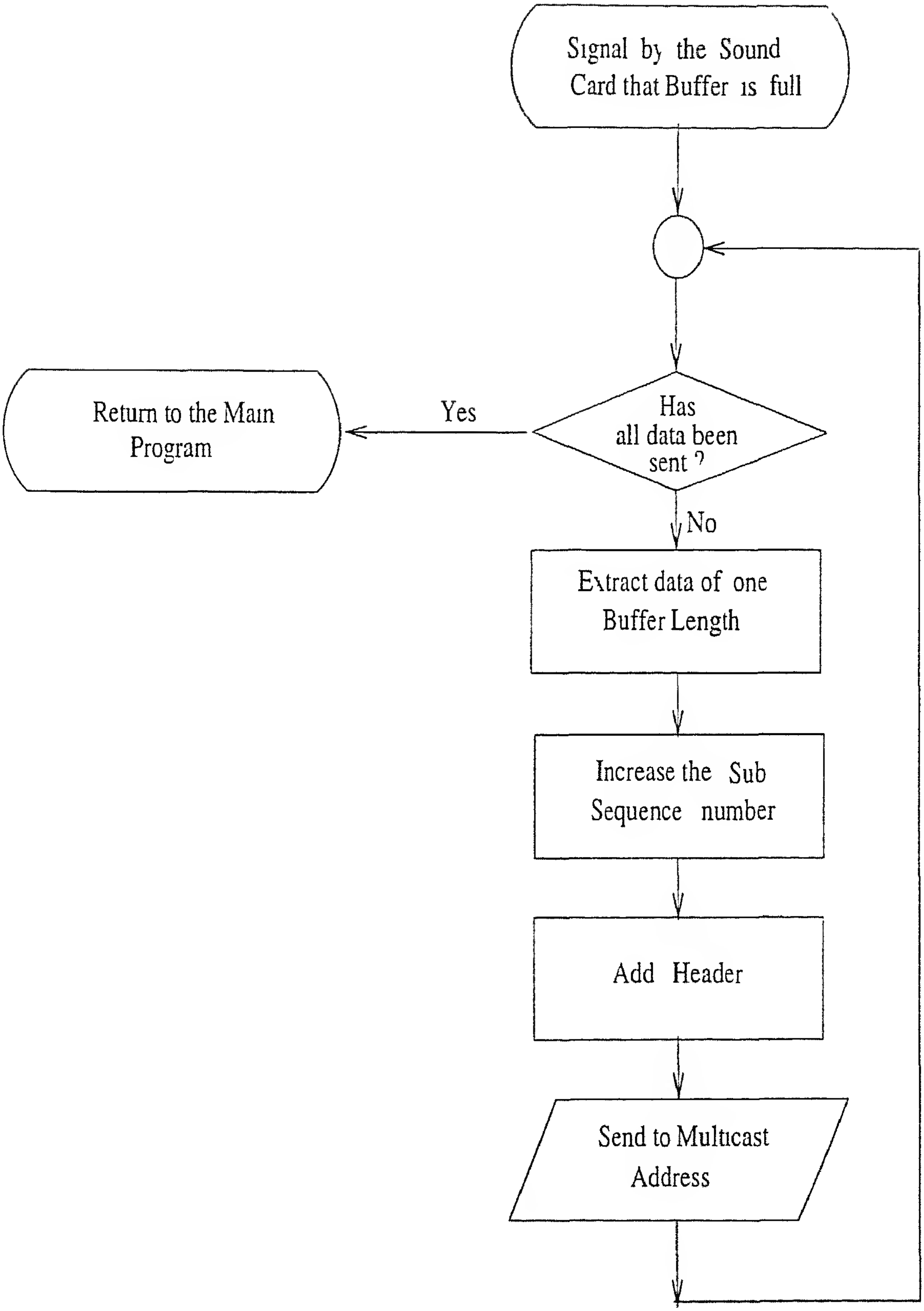

Fıgure 3 2 The Flow Dıagram for Packet Transmıssıon Scheme

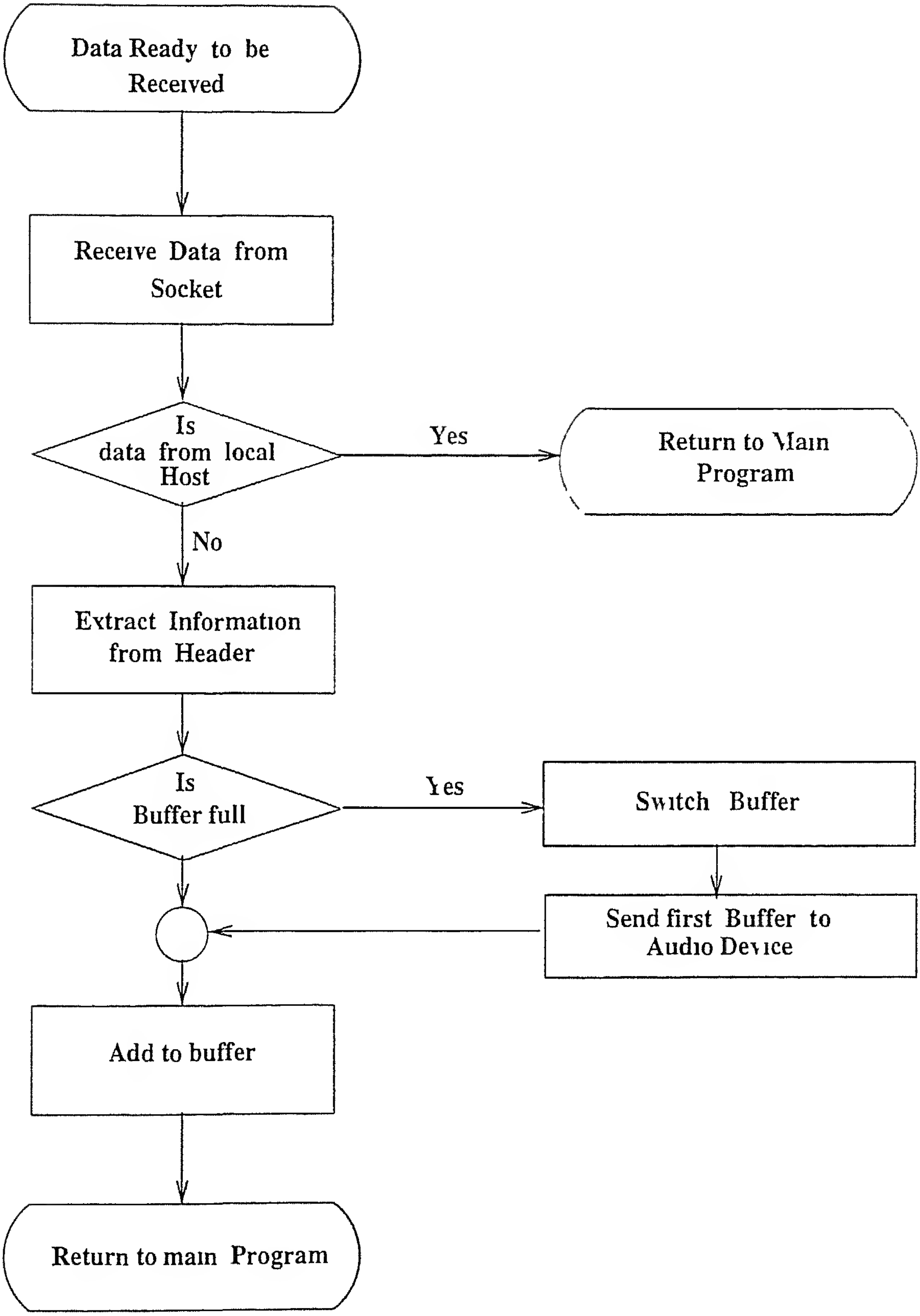

Figure 3 3 The Flow Diagram for Reception of Audio Data

In the case of packet loss or delay, the buffer which is used to store incoming packets may not yet be full when the other buffer's contents have been played out. In such a case, the two buffers form a queue to receive the incoming packets and on reception of the buffer once again switch to their normal roles. However, a jitter may be observed by the receiver.

The data which is transmitted can easily be sent over a WAN (since the packet is transmitted using UDP/IP protocol) and can be received even by a system whose operating system has a different byte order architecture than that of the sending machine. This is achieved by writing the data in the network byte order in the transmitted packet, and on receiving the packet the data is converted into the local byte order of the receiving system.

To cater to our requirement and to minimize the network traffic, we have chosen two sizes of SlideTalk packets. The first is 64 bytes - the packet for sending graphics and text information - it contains the required data and control information to recreate the objects. The other size is 1028 bytes - the packet for sending the voice data; in this packet the last 1000 bytes contain the voice data and the initial bytes contain the control information.

For synchronization of the packets received, for example, on a LAN connected to the transmitting LAN via a WAN where the packets can be received out of order, or the packets of a previous session can also be received, the header control information is used. The header contains the IP address of the sender and the process ID (which would differ for each conference initiated by this speaker). In the same session, when the speaker speaks at a different time a different sequence number is alloted each time, and all the packets getting generated are allotted monotonically increasing sub-sequence numbers. With the help of the sequence number and sub-sequence numbers the voice can be played out in smooth order at the receiver's end because the receiving station will drop the packet if its sub-sequence number is less than that of the one being played out.

We also looked into the aspect of multicasting the audio files' contents. The wave files start with a WAVEHDR which informs about the size of the file. First the WAVEHDR is sent and then the rest of the data is packetized and sent. At the receiver end the same information is put together once again in a wave file and then played out.

3.2.1 Talker Indication:

Since the delays between packet generation and playout can be substantial, it is inappropriate to indicate the talker at the time of packet arrival at the receiver end. In our case, for example, we are releasing our first data packet after an interval of 100 milliseconds; if the talker indication is received by the other members at that time, then by that time few other members might also have pressed their audio menu, presuming that they are the only ones to initiate the conference.

To avoid such a case, we send a control packet as soon as any one of the group members opts to speak. This packet informs the other members that soon the voice data shall be arriving. The control packet deactivates the speech menu of the receivers and sends a control signal to the receiving station to open its WAVE device in the playout mode. The device allocates sufficient buffers to deal with the incoming data. Similarly, when the speaker finishes the talk a control packet is released which informs the group members about it and activates their voice menu once again.

3.2.2 User Interface:

The user interface presented is very similar to any standard application running under MS Windows. We provide a menu bar with options to select drawing and text parameters like pen thickness, font and colour; audio option can be selected by pressing the appropriate menu switch. The hot keys for the menu items have also been provided. There is also a tool bar that presents an icon-driven interface to these operations.

3.2.3 Hardware and Software Requirements

The system requirements are as follows:

1. IBM compatible PC

2. MS-Windows 3.11 (WFWG)

3 Sound Blaster card

4 GhostView for Windows

5 Winsock 1 1

6 Media Control Interface

The SlideTalk software is available for Beta testing from Telematics lab[1] IIT Kanpur

In this thesis work thus, we have implemented the Voice Conferencing component of SlideTalk The communication mode during the conference is FIFO (the first person opting to speak will get the speech control and the microphone of the other members will get deactivated) We have used double buffer method for transmission as well as reception of the voice data This model of SlideTalk will fit well for a tele seminaring session where mainly one speaker will be active at a time However through an interrupt the audience can request the speaker to pass the speech control

[1] contact dmanju@iitk ernet in or svs@iitk ernet in

Chapter 4

Testing of the Implementation, and Conclusions

The SlideTalk in its present state of development supports exchange of voice data and slide file information along with graphics and text on a multicast framework The present framework however works under certain limitations imposed by the CPU speed and underlying network bandwidth

The following section discusses the testing of the implementation Section 4 2 contains concluding remarks and mentions the scope for future work

4 1 Testing of the implementation

In this thesis work the speech component of the SlideTalk has been implemented The implementation however works under two limitations The first limitation is imposed by the present CPU speed As mentioned in the previous chapter the CPU is intimately involved in the buffers' supply to the audio hardware their transmission and acquisition and their recording and play out Thus CPU is heavily loaded during real time voice conferencing Due to this reason we can have the communication mode as FIFO only and not free mode (in which the number of participants discussing simultaneously is not a constraint) In the free mode the CPU, with its present speed would not be able to process the received data if the number of participants increases to more than 2

The solution to avert such a problem can be a silence deletion algorithm such that no data be sent if a member is silent By using such an algorithm the load on CPU will come down drastically and would enable members to confer in the free mode

The second limitation is quite related to the first one This limitation is imposed by the underlying network with no priority for real time data On such a network even if the CPU is capable of handling free mode the amount of data generated in the free mode would require a preferred treatment (priority) from the underlying network so as to avoid jitter and minimize the discontinuities during real time conferencing Once again a solution like silence deletion algorithm will be a partial solution to this limitation

This model can be best utilized during a tele seminaring session with a speaker enabled to talk and the audience able to listen in real time

4 2 Conclusions and Scope for Future Work

The SlideTalk maintains the layered structure of the WhiteBoard Each layer has been assigned a specific function to perform

The sound component of the SlideTalk enables multicast of the speaker s voice to all the group members in real time The trickiest thing in the real time audio conferencing is the transmission and reception of the audio buffers We used a double buffering method for packet transmission as well as reception In such a scheme while one buffer is transmitted or received the other buffer remains with the audio hardware which fills in the voice data or plays out its contents The two buffers keep switching their roles Using such a scheme we have been able to run our sessions for quite long times (more than two hours) without any perceptible loss of synchronization

The SlideTalk with its present features of slide and sound, along with graphical annotating objects is an excellent tool for tele seminaring The present framework could be easily used to incorporate a real time video conferencing feature, using a similar technique as in sound to transmit and receive the video buffers An information server can also be provided to enable members to retrieve the information even if they join late or after the seminar session has ended A silence deletion algorithm can also be incorporated to enable free mode of communication during audio conference

References

[1] Manoj Gupta SlideTalk Slide Display Implementation *IIT Kanpur* 1997

[2] Vikas Gupta Raja Mukhopadhyay A Reliable Multicast Framework for Application *B Tech Project Report IIT Kanpur* 1996

[3] Floyd S Jacobson V McCanne S Liu C and Zhang L A Reliable Multicast Framework for Light Weight Sessions and Application Level Framing *SIGCOMM* 1995

[4] W Richard Stevens UNIX Network Programming *Prentice Hall of India* 1995

[5] Douglas E Comer Internetworking with TCP/IP Vol I *Prentice Hall of India* 1991

[6] Martin Hall Mark Towfiq Geoff Arnold and Tradewell Windows Socket An Open Interface for Network Programming under Microsoft Windows *Microsoft* 1992

[7] Jim Conger Windows Programming Primer Plus *Galgotia Publication*

[8] Reference Guide Borland Object Windows for C++ version 2 0 *Borland International Inc* 1993

[9] Programmer s Guide Borland Object Windows for C++ version 2 0 *Borland International Inc* 1993

[10] Tom Yager The Multimedia Production Handbook *Academic Press Professional* 1993

[11] Zhao Dai Aliasghar Babadi Kankanahallis MONET A Multimedia System for Conferencing and Application Sharing in Distributed System *Concurrent Engineering Research Center & Computer Science Dept West Virginia University* A Report

[12] Jonathan Rosenberg Gil Cruz Thomas Judd Presenting Multimedia Documents Over a Digital Network *Bellcore Morristown NJ US* 1 October 1991

[13] K Jeffay D L Stone F D Smith Transport and Display Mechanisms for Multimedia Conferencing Across Packet-Switched Networks *available at jeffay stone smithfd@cs unc edu* October 1992

[14] John Reidl Vahid Mashayekhi James Schnepf Mark Claypool and Dan Frankowski Suite Sound A System for Distributed Collaborative Multimedia *IEEE Transactions on Knowledge and Data Engineering* August 1993 pp 600 609

[15] Heinrich J Stuttgen Network Evolution and Multimedia Communication *IEEE Multimedia* Fall 1995 pp 42 58

[16] Eve M Schooler Case Study Multimedia Conference Control in a Packet switched Teleconferencing System *Internetworking Research and Experience Vol 4* 1993